A LINEAR-TIME ALGORITHM TO COMPUTE GEODESICS IN SOLVABLE BAUMSLAG-SOLITAR GROUPS

MURRAY ELDER

ABSTRACT. We present an algorithm to convert a word of length n in the standard generators of the solvable Baumslag-Solitar group $BS(1,p)$ into a geodesic word, which runs in linear time and space on a random access machine.

1. INTRODUCTION

Recently Miasnikov, Roman'kov, Ushakov and Vershik [5] proved that for free metabelian groups with standard generating sets, the following problem is NP-complete:

- given a word in the generators and an integer k, decide whether the geodesic length of the word is less than k.

They call this the *bounded geodesic length problem* for a group G with finite generating set $\mathcal{G}$. It follows that given a word, computing its geodesic length, and finding an explicit geodesic representative for it, are NP-hard problems. These problems are referred to as the *geodesic length problem* and the *geodesic problem* respectively.

In this article we consider the same problems for a different class of metabelian groups, the well known *Baumslag-Solitar groups*, with presentations

$$\langle a, t \mid tat^{-1} = a^p \rangle$$

for any integer $p \geq 2$. We give a deterministic algorithm which takes as input a word in the generators $a^{\pm 1}, t^{\pm 1}$ of length n, and outputs a geodesic word representing the same group element, in time $O(n)$. Consequently, the three problems are solvable in linear time[1].

In an unpublished preprint [6] Miller gives an effective procedure to convert a word in the above group to a geodesic of the form $t^{-k}zt^{-m}$ where z belongs to a regular language over the alphabet $\{a, a^{-1}, t\}$. Miller's algorithm as it is

Date: February 9, 2010.

2000 *Mathematics Subject Classification.* 20F65, 68Q25.

Key words and phrases. Baumslag-Solitar group, metabelian group, solvable group, linear time algorithm, geodesic.

[1]It is clear that solving the geodesic problem implies the other two. In [2] the author and Rechnitzer show they are all in fact equivalent

given in his (unpublished) preprint would take exponential time worst case. The algorithm presented here closely follows Miller's procedure with some modifications (using pointers in part one and a careful tracking procedure in part two) to ensure linear time and space. We use as our computational model a *random access machine*, which allows us to access (read, write and delete) any specified location in an array in constant time.

Recent work of Diekert and Laun [1] extends the result of this paper to groups of the form $\langle a, t \mid ta^p t^{-1} = a^q \rangle$ when p divides q. Their algorithm runs in quadratic time, but in the case $p = 1$ the time reduces to linear.

The author wishes to thank Sasha Ushakov and Alexei Miasnikov for suggesting this problem, and Sasha, Alexei, Alex Myasnikov, Igor Lysionok, Andrew Rechnitzer and Yves Stalder for many helpful suggestions. The author thanks the anonymous reviewer for their careful reading and instructive comments and corrections. The author gratefully acknowledges the support of the Algebraic Cryptography Center at Stevens Institute of Technology.

2. Preliminaries

Fix G_p to be the Baumslag-Solitar group $\langle a, t \mid t^{-1}at = a^p \rangle$ for some $p \geq 2$. We will call a single relator a *brick*, with sides labeled by t edges, as in Figure 1. The Cayley graph can be obtained by gluing together these bricks. We call a *sheet* a subset of the Cayley graph made by laying rows of bricks atop each other to make a plane, also shown in Figure 1. The complete Cayley graph is

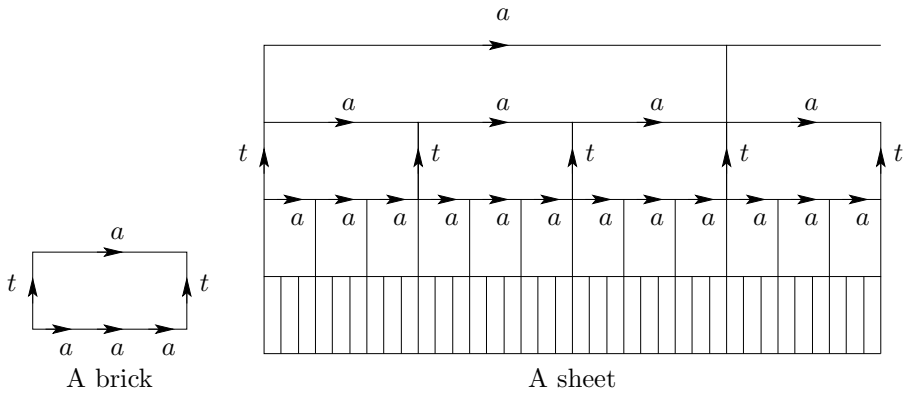

Figure 1. Parts of the Cayley graph of G_3

obtained by gluing these sheets together so that every vertex has degree four. From side-on the Cayley graph looks like a rooted p-ary tree. We orient each a edge to the right and each t each going up. Some nice pictures can be found in [3] pp. 155-160.

A word in the generators $a^{\pm 1}, t^{\pm 1}$ is said to be of the form P if it contains no t^{-1} letters and at least one t letter, and of the form N if it contains no t's and at least one t^{-1}. Then a word is of the form PNP say, if it is the concatenation of three words of the form P, N, P in that order. The t-exponent sum of a word is the number of t letters minus the number of t^{-1} letters. We write $=_G$ when two words represent the same element in the group and $=$ when they are identical as strings, and $\ell(w)$ is the number of letters in the string w.

The following simple lemma and corollaries come from [6] and [4].

Lemma 1 (Commutation)**.** *If u, v have t-exponent sum zero then $uv =_G vu$.*

Corollary 2 (Geodesics)**.** *A geodesic cannot contain a subword of the form $NPNP$ or $PNPN$.*

Corollary 3 (Pushing as)**.** *If w is of type NP and has t-exponent zero then $w =_G u = t^{-k}u_P$ where u_P is of type P and t-exponent k, and $\ell(u) \leq \ell(w)$. If w is of type PN and t-exponent zero then $w =_G v = v_P t^{-k}$ where v_P is of type P and t-exponent k, and $\ell(v) \leq \ell(w)$.*

The two corollaries are simply a matter of commuting subwords of t-exponent zero past each other. We will show how this can be done in linear time and space in the algorithm. The trick is to use *pointers*, which we will explain in Section 4 below.

Here is an outline of the algorithm. We start with an input word of length n, which we write onto a tape of length n, freely reducing as we write. Scanning through the word and keeping track of the t-exponent we can determine the total t-exponent sum of the input word. If it is negative, write the inverse of the word on a new tape (which takes linear time), and remember to output the inverse of the final result at the end. So without loss of generality our input word has non-negative t-exponent sum.

The first part of the algorithm transforms the input word into a word of the form $u = t^{-k}u_P t^{-m}$, as Miller does, which is equal in the group to the input word, where the word $u_P = a^{\epsilon_0} t \dots t a^{\epsilon_q}$ with $q \geq k + m$, $|\epsilon_i| < p$ for $i = 0, \dots, q-1$ and $|\epsilon_q| < 3p$. This word is not necessarily geodesic but is close. In part two of the algorithm we prove that u is (almost) fellow traveled by a geodesic for it, which lives in the same sheet of the Cayley graph, and because of this fact we are able to trace along u and find a geodesic for it in linear time and space.

3. Data structure and subroutines

We will make use of the following data structure and subroutines in part one of the algorithm. Assume the input word has length n.

Construct a list we will call List A of $n + 2$ 5-tuples, which we view as an $5 \times (n+2)$ table. Each *address* in the table will contain either a blank symbol,

an integer (between $-n$ and $n+1$), or the symbol t, t^{-1}, a, a^{-1}, S or F. We refer to an address by the ordered pair (row,column).

- Write the numbers 0 to $n+1$ in the first row. These entries will stay fixed throughout the algorithm.
- Row 2 will store the input word. Write S for *start* at address (2,0), then the input word letter by letter in addresses $(2,1)$ to $(2,n)$, and at address $(2,n+1)$ write F for *finish*. As the algorithm progresses, these entries will either remain in their original positions, or be erased (and replaced by a blank symbol). S and F are never erased.
- Row 3 will contain no entries at the beginning. As the algorithm progresses we will use the addresses in this row to store integers (between $-n$ to n).
- Write the numbers 1 through $n+1$ in the first $n+1$ addresses of row 4. Leave the final address blank. This row will act as a *pointer* to the next column address to be read. As the algorithm progresses, the entries in this row may change.
- In row 5, write a blank symbol in the first address, then write the numbers 0 through n in the remaining addresses. This row indicates the previous column address that was read.

Here is List A in its initial state, with input word $at^2a \dots at^{-1}$.

List A	$\downarrow$								
column	0	1	2	3	4	$\cdots$	$n-1$	n	$n+1$
word	S	a	t	t	a	$\cdots$	a	t^{-1}	F
$t-$ exp									
to	1	2	3	4	5	$\cdots$	n	$n+1$	
from		0	1	2	3	$\cdots$	$n-2$	$n-1$	n

As the algorithm progresses, we will "reorder" the word written in row 2 using the pointers in rows 4 and 5 (and leaving the letters in row 2 fixed, possibly erasing some). To read the word, start at the S symbol. Move to the column address indicated in row 4. At the beginning this will be column 1. From the current column read the entry in row 4 to move the the next column. Continue until you reach the F symbol. At any stage, to step back to the previous address, go to the column address indicated by row 5. Throughout the algorithm, the pointers will never point to or from a column which has a blank symbol in row 2. The pointers allow us to rearrange and delete letters from the word in row 2 efficiently (in constant time), without having to move any letters on the table.

For convenience, we indicate the current address being read by a *cursor*. We assume that moving the cursor from one position in the list to another takes constant time on a random access machine.

Here are two subroutines that we will use many times. Each one takes constant time to call. Assume that the cursor is pointing to column k, and that the entry in the address $(2, k)$ is not blank.

Subroutine 1: Free reduction

Read the entries in rows 2,4 and 5 of column k. Say the letter in row 2 is x, and the integers in rows 4 and 5 are i, j.

If position j row 2 is x^{-1}, then we can cancel this pair of generators from the word as follows:

- read the integer in row 5 position j, and go to the address indicated (say it is r). In row 4 of this address, write i. In row 5 position i, write r.
- erase entries in columns j and k
- go to position i.

In this way, we have deleted $x^{-1}x$ from the word, and adjusted the pointers so that they skip these positions.

List A						$\downarrow$			
column	...	r	...	j	...	k	...	i	...
word				x^{-1}		x			
$t-$exp									
to						i			
from				r		j			

$\rightsquigarrow$

List A								$\downarrow$	
column	...	r	...	j	...	k	...	i	...
word				$\times$		$\times$			
$t-$exp				$\times$		$\times$			
to		i		$\times$		$\times$			
from				$\times$		$\times$		r	

Else, if position i row 2 is x^{-1}, we perform a similar operation to erase xx^{-1} from the word, adjusting pointers appropriately.

Assuming that we can access positions using the pointers in constant time (that is, we have a random access machine), then this procedure takes constant time to run.

The next subroutine eliminates the occurrence of subwords $a^{\pm 3p}$ (where $p \geq 2$ is fixed). Again, assume the pointer is at position k and the entry at address $(2,k)$ is not blank.

Subroutine 2: Consecutive as

- If the letter at this address is a, set a counter `acount` $=$ `1`. Move back one square (using pointer in row 5) to column j. If address $(2,j)$ is a, increment `acount`. Repeat until `acount` $=$ `3p` or the next letter is not a. Note maximum number of steps is $3p$ (constant).

If `acount` = `3p` and you are at column i, write ta^3t^{-1} over the first 5 as, and blank symbols in the remaining $3p-5$ addresses up to position k. Adjust the pointers so that the pointer at the added t^{-1} points to the value indicated at (4,k), and write the appropriate value in row 5 of that position.

- If the letter at this position is a^{-1}, do the same with a^{-1} instead of a.

This procedure takes constant time, and if it is *successful* (that is, replaces $a^{\pm 3p}$ by $t^{-1}a^{\pm 3}t$) it strictly reduces the length of the word.

4. Algorithm part 1

Assume the input word has length n.

Step 1: Write the input word in freely reduced form on List A as follows. Read the first letter and write it in address $(1, 2)$ of List A. For each subsequent letter if it freely cancels with the previous letter in row 2, erase the previous letter and continue. At the same time record the successive t-exponent sum of the word by incrementing and decrementing a counter each time a $t^{\pm 1}$ is read. If the final t-exponent sum is negative, rewrite the inverse of the word on a new List A. At the end of the algorithm, remember to output the inverse of the final output.

So the word in row 2 of the tape is freely reduced and has nonnegative t-exponent. Fill in rows 1,4 and 5 of List A with the column numbers and pointers set to the initial state.

Step 2: Move cursor to column 0, then take $3p - 1$ steps to the right to column k. Assume the entire word in row 2 is freely reduced, and the word up to column k contains no more than $3p - 1$ consecutive as or a^{-1}s. Go to the next column (indicated by row 4), and perform Subroutine 2.

If the subroutine finds $a^{\pm 3p}$, then with the cursor at the each end of the inserted word (of length 5), perform Subroutine 1. Repeat until Subroutine 1 finds no more canceling pairs. Then move back to the next position after k and continue applying Subroutine 2.

At the end of this procedure, the entire word is freely reduced and has no $a^{\pm 3p}$ subwords. The number of times Subroutine 2 is performed is at most n, and the total number of times we perform Subroutine 1 is $O(n)$ since each time it is successful the word reduces length, so it is successful at most n times and unsuccessful at most twice (for each end) after each application of Subroutine 2.

So we now have a freely reduced word with less than $3p$ $a^{\pm 1}$ letters in succession, in row 2 of List A. The pointers in row 4 still point to columns to the right, since we have not commuted any subwords yet.

Step 3: Construct a second list we call List B of $2n+1$ 4-tuples, which we view as a $4 \times (2n+1)$ table. In the first row write the integers from $-n$ to n.

Starting at column 0 of List A, set a counter `texp`=0. Reading the word in row 2 from left to right, if in column k you read a $t^{\pm 1}$ letter, add ± 1 to `texp`, and write the value of `texp` at address $(3, k)$ of List A. In List B, if address (2,`texp`) is blank, write k. If it contains a value, write k in address (3,`texp`) if it is blank, otherwise in address (4,`texp`).

In other words, each time you read a $t^{\pm 1}$, write the current t-exponent sum underneath it, and in List B keep a record of how many times this t-exponent has appeared (which we call the number of *strikes* for that exponent) and at which positions in List A it appeared.

Here we show List A for the input word $at^2atat^{-1}at^{-1}at^{-1}ata\ldots$, and the corresponding List B, as an example.

List A														↓		
column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	$\cdots$
word	S	a	t	t	a	t	a	t^{-1}	a	t^{-1}	a	t^{-1}	a	t	a	
$t-$exp			1	2		3		2		1		0		1		
to	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
from		0	1	2	3	4	5	6	7	8	9	10	11	12	13	

List B															
$t-$exp	$\cdots$	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	$\cdots$
strike 1								11	2	3	5				
strike 2									9	7					
strike 3									13						

When an entry occurs in the last row of List B at some position labeled `texp`, meaning some exponent has has "3 strikes", then we have a prefix of the form either $NPNP$ or $PNPN$, so we apply Corollary 2 as follows. Suppose the entries in this column are p_a, p_b, p_3, with p_3 the most recently added. These correspond to the positions in List A where the value `texp` have appeared.

To begin with, the word written in row 2 of List A appears in the correct order (from left to right), the pointers have only been used to possibly skip blank addresses. So at the start of this step we know that p_a comes before p_b. As the algorithm progresses, we will not know which of p_a and p_b comes first in the word. That is, as we introduce pointers to List A to move subwords around, a letter in column p could sit before a letter in column q with $q < p$. We do know that p_3 is the right-most position.

The word read in its current order is either $\ldots p_a \ldots p_b \ldots p_3 \ldots$ or $\ldots p_b \ldots p_a \ldots p_3 \ldots$. We can determine the order with the following subroutine.

Subroutine 3: Determine order of p_a, p_b.

Starting at p_a, scan back (using pointers in row 5) through the word to the position of the previous $t^{\pm 1}$ letter, or the S symbol. Since we have at most $3p-1$ consecutive $a^{\pm 1}$ letters, this takes constant time. Do the same for p_b.

If we come to S from either p_a or p_b, then we know that this position must come first.

If both p_a, p_b are preceded by $t^{\pm 1}$ letters, then we need more information. Start at p_a and scan forward to the first $t^{\pm 1}$, whose position we call q_a. Start at p_b and scan forward to the first $t^{\pm 1}$, call this position q_b. This takes constant time since there are at most $3p-1$ consecutive $a^{\pm 1}$ letters.

Now one of q_a, q_b must contain a $t^{\pm 1}$ in row 2, with sign opposite to that of p_3.

- If q_a is same sign as p_3, then order must be $p_a - q_a - p_b - q_b - p_3$
- If q_b is same sign as p_3, then order must be $p_b - q_b - p_a - q_a - p_3$
- Both q_a, q_b have opposite sign to p_3. In this case, we look at the letters in row 2 positions p_a, p_b. If p_a has opposite sign to p_3, then it must come first, since one of p_a, p_b must match up with q_a, q_b.

  If both p_a, p_b have same letter as p_3 in row 2, then we are in a situation like $tat^{-1}atat^{-1}at$. But since there is a $t^{\pm 1}$ letter preceding both p_a and p_b, then the t-exponent before p_a, p_b, p_3 are read is the same, and is recorded three times. This case cannot arise since we apply this procedure the first time we see the same number more than twice.

Using this subroutine we can determine the correct order of the positions p_a, p_b and q_a, q_b, in $O(n)$ time. Rename the first position p_1 and second p_2, and q_1, q_2 as appropriate. So we have $p_1 - q_1 - p_2 - q_2 - p_3$

The subword between positions q_1 and p_2 has t-exponent 0, as does the subword from q_2 to p_3. By commuting one of these subwords (using Lemma 1) we can place a t next to a t^{-1} somewhere and get a free cancellation. The precise instruction will depend on the letters at each of these positions, and we will consider each situation case-by-case.

Case 1:

List A											
column	$\cdots$	p_1	$\cdots$	q_1	$\cdots$	p_2	$\cdots$	q_2	$\cdots$	p_3	$\cdots$
word		t		t^{-1}		t		t^{-1}		t	
$t-$exp		k_1		k_2							
to		i_1		i_2		i_3					
from		j_1		j_2		j_3					

Between p_1 and q_1 we have only a letters (or nothing). So we will commute the subword $q_1 - p_2$ back towards p_1 as follows:

- j_1 row 4, replace p_1 by i_2
- i_2 row 5, replace q_1 by j_1

- p_2 row 4, replace i_3 by i_1
- i_1 row 5, replace p_1 by p_2
- j_2 row 4, replace q_1 by i_3
- i_3 row 5, replace p_2 by j_2
- delete columns p_1, q_1
- delete p_1 and q_1 from List B columns k_1 and k_2 respectively.

This has the effect of moving the subword back through the word, but without changing more that a constant number of entries in the lists (with random access).

The next case applies to our running example shown above.

Case 2:

List A											
column	$\cdots$	p_1	$\cdots$	q_1	$\cdots$	p_2	$\cdots$	q_2	$\cdots$	p_3	$\cdots$
word		t		t		t^{-1}		t^{-1}		t	
$t-$exp		k_1						k_2			
to		i_1						i_2		i_3	
from		j_1						j_2		j_3	

This time we will commute the subword $q_2 - p_3$ back past the subword of t-exponent zero and next to p_1 as follows:

- j_1 row 4, replace p_1 by i_2
- i_2 row 5, replace p_1 by j_1
- p_3 row 4, replace i_3 by i_1
- i_1 row 5, replace p_1 by p_3
- j_2 row 4, replace q_2 by i_3
- i_3 row 5, replace p_3 by j_2
- delete columns q_1, p_2
- delete p_1 and q_2 from List B columns k_1 and k_2 respectively.

Below we show the two lists after commuting and deleting tt^{-1} for our running example after this step. We read the new word off List A following the pointers as $a \rightsquigarrow at \rightsquigarrow tatat^{-1}at^{-1}a \rightsquigarrow at^{-1} \ldots$.

List A														$\downarrow$		
column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	$\cdots$
word	S	a	$\times$	t	a	t	a	t^{-1}	a	t^{-1}	a	$\times$	a	t	a	
$t-$exp			$\times$	2		3		2		1		$\times$		1		
to	1	12	$\times$	4	5	6	7	8	9	10	14	$\times$	13	3	15	
from		0	$\times$	13	3	4	5	6	7	8	9	$\times$	1	12	10	

List B															
$t-$exp	$\cdots$	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	$\cdots$
strike 1									9	3	5				
strike 2									13	7					
strike 3															

The remaining cases are similar and we leave it to the reader to imagine the instructions for each one. Corollary 2 guarantees that some commutation will reduce length in each case.

With the cursor at position p_3 or if blank, the non-blank letter to its right, perform Subroutine 1 until unsuccessful, then Subroutine 2 until unsuccessful, and alternately until both are unsuccessful. Since each successful application of a subroutine reduces word length, the total number of successes of each is n throughout the whole algorithm. Once both are unsuccessful the entire word is again freely reduced and avoids $a^{\pm 3p}$, and the cursor is at the next non-blank position to the right of p_3. We then resume Step 3 from this position.

So after performing this procedure, List A contains a possibly shorter word in row 2, which is read starting at column 0 and following pointers, and List B contains the correct data of t-exponents and positions (although positions don't stay in order). Since we removed one of the 3 strikes, we start at $p_3 + 1$ and continue filling out row 3 of List A, adding appropriate entries to List B, until we again get 3 strikes. Note that we do not backtrack, so the total number of right steps taken in Step 3 (assuming the random access model of computation allows us to read and write at any specified position in the table) is $O(n)$. The number of times we need to apply the subroutines (successfully and unsuccessfully) is also $O(n)$ regardless of how many times Step 3 is called, so so all together this step takes $O(n)$ time.

At the end of this step, since the word has nonnegative t-exponent sum, and all "3 strikes" have been eliminated, the word must be of the form E, P, PN, NP, NPN, or PNP.

Step 4: If the word at this stage is of the form PNP, its t-exponent sum must be zero, so it has a prefix of the form PN and suffix NP, both of zero t-exponent sum. Say p_1, p_2 are the two positions that the t-exponent is zero. We commute the prefix and suffix by rewriting pointers. If the word on List A is not written in order from left to right, we can create a new List A in which the word is in correct order, by reading the current list following the pointers.

So after this we can assume the configuration of List A is as follows:

List A							
column	0	$\cdots$	p_1	$\cdots$	p_2	$\cdots$	$n+1$
word	S		t^{-1}		t		F
$t-$ exp			0		0		
to	i_1		i_2		i_3		
from			j_1		j_2		j_3

Then do the following:

- 0 row 4 replace i_1 by p_2
- p_2 row 5 replace j_2 by 0
- j_3 row 4 replace $n+1$ by p_1
- p_1 row 5 replace 0 by j_3
- j_1 row 4 replace p_2 by $n+1$
- $n+1$ row 5 replace j_3 by j_1.

The word is now the form NPN.

Step 5: At this point the word in List A row 2 is of the form E, P, PN, NP or NPN. We can ascertain which of these it is in constant time simply by checking the first and last $t^{\pm 1}$ letter in the word, which lie at most $3p$ steps from the ends of the tape (positions S and F), following pointers.

- No $t^{\pm 1}$ letters: E
- First t last t: P
- First t last t^{-1}: PN
- First t^{-1} last t: NP
- First t^{-1} last t^{-1}: NPN

In the case E, the word is a^i with $|i| < 3p$, so by checking a finite list we can find a geodesic for it and be done. So for the rest of the algorithm assume u is of the form P, PN, NP, NPN.

In the last two cases NP and NPN it is possible the word contains a subword of the form $t^{-1}a^{xp}t$ for an integer x, which will be in $\{\pm 1, \pm 2\}$. If so we want to replace it by a^x, which will always reduce length. This is easily done in constant time, assuming the cursor is pointing to the column containing the first t letter in the word. From this column scan back at most $3p$ letters to a t^{-1} letter and count the number of $a^{\pm 1}$ letters in between. Then if you find $t^{-1}a^{xp}t$ rewrite with a^x and then move forward to the next t letter. Repeat this at most $O(n)$ times (the maximum number of t letters in the word) until no such subword appears. Note that to locate the first t letter in the word to start this takes $O(n)$ time to scan the word, but you only need to do this once.

We now have a word in one of the forms P, PN, NP, NPN which contains no $t^{-1}a^{xp}t$ subword.

Step 6: In this step we apply Corollary 3 to push all of the $a^{\pm 1}$ letters in the word occur in a single sheet of the Cayley graph. The output of this step (the word in row 2 of List A) will be a word of the form $t^{-k}u_P t^{-l}$ where u_P is a word of type P with t-exponent $\geq k+l$, and $k, l \geq 0$ and the word still does not contain a subword of the form $t^{-1}a^{xp}t$.
Case P**:** The word is of the form u_P so done.
Case PN**:** Say `texp` is the final t-exponent of the word, which occurs at positions p_1 and q_1. If q_1 is not the end of the word (that is, there are $a^{\pm 1}$ letters at the end of the word), then we want to push the a letters there back through the word to t-exponent 0, which starts after p_1. The configuration of the tape is as follows (where we assume $j_3 \neq q_1$ since there are $a^{\pm 1}$ letters at the end of the word):

List A						
column	$\cdots$	p_1	$\cdots$	p_2	$\cdots$	$n+1$
word		t		t^{-1}		F
$t-$exp		`texp`		`texp`		
to		i_1		i_2		
from		j_1		j_2		j_3

Then do the following:

- q_1 row 4, replace i_2 with $n+1$
- $n+1$ row 5, replace j_3 with q_1
- p_1 row 4, replace i_1 with i_2
- i_2 row 5, replace q_1 with p_1
- j_3 row 4, replace $n+1$ with i_1
- i_1 row 5, replace p_1 with j_3

So we have commuted the word $a^{\pm m}$ at the end, through the subword of t-exponent 0. Check for cancellation of $a^{\pm 1}a^{\mp 1}$, if this occurs then cancel. Repeat up to $3p-1$ (constant) times.

Next, let p_2, q_2 be the positions that (`texp` $+1$) occurs. Repeat the procedure at this level. Again freely cancel.

Iterate this until all $a^{\pm 1}$ letters are pushed into the middle of the word, so the resulting word is of the form $u_P t^{-k}$ where u_P is a word of type P with t-exponent at least k. Note that at the top level there is only one a^i subword, which will not be canceled, so there is no free cancellation of t letters in this step. At every other level there can be at most $6p-2$ consecutive $a^{\pm 1}$ letters.
Case NP**:** Same as previous case, this time pushing as to the right.
Case NPN**:** Break the word into NP and PN subwords, with the NP subword ending with t, and each of zero t-exponent sum, and perform the above steps to push a letters to the right and left respectively, then ensure there is no $t^{-1}a^{ip}t$ subword. In the NP prefix the maximum number of consecutive $a^{\pm 1}$ letters is $6p-2$ at every level except the top level, since we

cut the word immediately after a t, so at this level we have at most $3p-1$ consecutive $a^{\pm 1}$s, and in the PN suffix we can have at most $6p-2$ at every level, so all together there could be at most $9p-3$ consecutive $a^{\pm 1}$s.

So after this step, the positive part of the word, u_P, stays within a single sheet of the Cayley graph. We write

$$u_P = a^{\epsilon_0} t a^{\epsilon_1} \dots a^{\epsilon_{m-1}} t a^{\epsilon_m}$$

where each $|\epsilon_i| < 9p$ and with ϵ_0 not a multiple of p when the word is of the form NP or NPN.

Step 7: In this step we remove all occurrences of $a^{\pm p}t$ in the word. Scan to the first t in row 2. If the preceding p letters are $a^{\pm 1}$ then replace $a^{\pm p}t$ by $ta^{\pm 1}$. Stay at this t letter and repeat until there is no $a^{\pm p}t$, then move to the next t letter. Since each replacement reduces length the time for this step is linear. At the end you have eliminated all $a^{\pm p}t$ subwords so the word is of the form $u = t^{-k}u_P t^{-l} = t^{-k}a^{\epsilon_0}t \dots t a^{\epsilon_m} t^{-l}$ with $|\epsilon_i| < p$ for $i < m$.

Step 8: Set $\epsilon_m = M_0$. If $|M_0| < 3p$ then stop, part one of the algorithm is done. If $|M_0| \geq 3p$ then we will replace the last term $a^{\epsilon_m} = a^{M_0}$, by a word of the form $a^{\eta_0} t \dots t a^{\eta_s} t^{-s}$ with $|\eta_i| < p$ for $i < s$ and $|\eta_s| < 3p$, as follows.

- Go to the first t^{-1} after a^{M_0} on the tape, then scan back $3p$ steps. If you read $a^{\pm 3p}$ in these steps, then replace the subword by $ta^{\pm 3}t^{-1}$, which strictly reduces length. Then scan back another p steps from the t letter you have inserted, and if you read $a^{\pm p}t$ then replace it by $ta^{\pm 1}$. Repeat until you don't read p consecutive as or a^{-1}s.

  If you did any replacing, you now have a word of the form $u = t^{-k}a^{\epsilon_0}t \dots a^{\epsilon_{m-1}} t a^{\eta_0} t a^{M_1} t^{-1} t^{-l}$ with $|\epsilon_i| < p$ for $i < m$ and $|\eta_0| < p$. The number of steps to do this is $O(|M_0|)$. Note that $|M_1| \leq |M_0|/p$.
- Repeat the previous step, by scanning back $3p$ from the last t^{-1} inserted. If you read $a^{\pm 3p}$ replace and repeat the procedure as before, and if $|M_i| < 3p$ stop. Note that each $|M_i| \leq |M_{i-1}|/p$, so $|M_i| =\leq |M_0|/(p^i)$.

Each iteration of this takes $O(|M_i|) = O(|M_0|/(p^i))$ steps, so in total the time for this procedure is

$$O(|M_0| + |M_0|/p + |M_0|/p^2 + \dots) = O(|M_0|) = O(n)$$

by the geometric series formula.

So the word on the tape is now of the form $u = t^{-k}a^{\epsilon_0}t \dots a^{\epsilon_q}t^{-m}$ where $0 < |\epsilon_0| < p$ if $k > 0$, $|\epsilon_i| < p$ for $0 < i < q$ and $|\epsilon_q| < 3p$.

We summarise part one of the algorithm in the following lemma.

Lemma 4. *Any word $w \in G_p$ can be converted to a word $u =_G w$ of the form*

$$u = t^{-k}a^{\epsilon_0}t \dots t a^{\epsilon_q} t^{-m}$$

with $\ell(u) \leq \ell(w)$, $k, m, q \geq 0$, $q \geq k + m$,$0 < |\epsilon_0| < p$ if $k > 0$, $|\epsilon_i| < p$ for $i = 0, \dots q-1$ and $|\epsilon_q| < 3p$, and moreover this can be achieved in linear time and space in the length of w.

5. Part two of the algorithm

The key to the final part of the algorithm are the following two lemmas. Roughly, they say that now that we have put the input word into a nice enough form, a geodesic for it can be found which (almost) fellow travels it in the same sheets of the Cayley graph, and so a geodesic can be obtained by tracking our current word in linear time and space.

Lemma 5. *Let*

$$u = t^{-k} a^{\epsilon_0} t \dots t a^{\epsilon_q} t^{-m}$$

with $q \geq k + m$, $k, m \geq 0$, $\epsilon_0 \neq 0$ if $k > 0$, $|\epsilon_i| < p$ for $i = 0, \dots q-1$ and $|\epsilon_q| < 3p$. Then there is a geodesic v for u of the form

$$v = t^{-k} a^{\eta_0} t \dots t a^{\eta_r} t^{-s}$$

with $k, r, s \geq 0$, $\eta_0 \neq 0$ if $k > 0$, $q - m = r - s$, $|\eta_i| < p$ for $i = 0, \dots r-1$ and $|\epsilon_r| < 3p$.

Proof. Let v be some geodesic for u, then applying part one of the algorithm we can put v into the form $t^{-j} a^{\eta_0} t \dots t a^{\eta_r} t^{-s}$ for some integers j, r, s, and $\eta_0 \neq 0$ if $j > 0$, $|\eta_i| < p$ for $i = 0, \dots r-1$ and $|\epsilon_r| < 3p$, otherwise v is not geodesic, and $q - k - m = r - j - s \geq 0$ since $u =_G v$.

Apply the substitutions $t a^i t^{-1} = a^{ip}$ to u m times to get $u' = t^{-k} u_1 a^c$ where u_1 is a prefix of $a^{\epsilon_0} t \dots t a^{\epsilon_q}$ ending in t, of t-exponent sum $q - m$, and c some integer. Do the same to v to get $v' = t^{-j} v_1 a^d$ where v_1 is a prefix of $a^{\eta_0} t \dots t a^{\eta_r} t$.

Since $u =_G v\ 1 =_G v^{-1} u =_G a^{-d} v_1^{-1} t^{j-k} u_1 a^c$ which is of the form NP. By Britton's Lemma this word contains either $t a^i t^{-1}$ or $t^{-1} a^{ip} t$, but since it is of type NP the subword must be $t^{-1} a^{ip} t$. If $j > k$ this means η_0 is a multiple of p, and if $k > j$ then ϵ_0 is a multiple of p, either way a contradiction. So $k = j$.

This means that u and v both have the form $t^{-k} PN$. □

We now complete the algorithm by proving that we can find such a v in the same sheet as u in linear time and space in the length of u. For ease of exposition we divide this into two cases. We first consider the case that u is of the form P or NP.

Lemma 6. *Let u be as in the previous lemma with $m = 0$, that is, $u = t^{-k} a^{\epsilon_0} t \dots t a^{\epsilon_q}$. Then v can be found, as in the previous lemma, so that u, v asynchronously $3p$-fellow travel up to level $q - m$, in linear time and space in the length of u.*

Proof. We draw a neighborhood of u in the Cayley graph as follows. Start at the endpoint of t^{-k}, and label this point S. Draw a horizontal line of $|\epsilon_0|$ a edges, left if $\epsilon_i < 0$ and right if positive. Then draw a vertical t edge up from this line, and complete the picture by drawing in the brick containing $a^{\epsilon_0}t$ on its boundary. Label the corner corresponding to the endpoint of $a^{\epsilon_0}t$ by U, and on each corner compute the distance d_1, d_2 back to S (which will be $|\epsilon_0| + 1$ and $p - |\epsilon_0| + 1$). If $|d_1 - d_2| \leq 1$ then keep both labels, and store two words g_1, g_2 which are geodesics to these points. If their difference is greater than 1 then discard the larger label and only keep the short one, plus a geodesic word g_1 to it. See Figure 2.

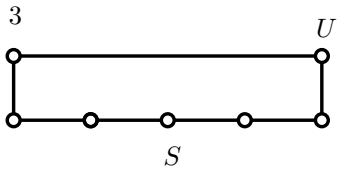

FIGURE 2. First level. In these figures we are in the group G_4.

Now assume you have drawn this picture up to level $i < q$, so you have i bricks stacked on top of each other vertically, and the top brick has its top corner(s) labeled U corresponding to the endpoint of $a^{\epsilon_0}t \dots a^{\epsilon_{i-1}}t$, and $d_1, (d_2)$ the shortest distance(s) back to S. Also you have stored geodesic(s) $g_1, (g_2)$ to the points labeled d_1, d_2.

From the point U, draw a horizontal line for a^{ϵ_i} to the left or right depending on the sign. Then draw a vertical t edge up. Now since $|\epsilon_i| < p$, the brick with boundary $a^{\epsilon_i}t$ also contains the point(s) $d_1, (d_2)$, and so to compute the distance to the corners of the new brick, one simply computes from these points, since they are the closest points on the level i in this sheet. So label the corners of the new brick in level $i + 1$ by $U, d_1, (d_2)$ as before. Update $g_1, (g_2)$ by appending suffix(es) $a^j t$.

In this way one can draw the path u in its sheet up to level q, and keep track of the distances from S to each level of the sheet, using constant time and space for each level. Figure 5 shows the next two iterations of this.

At level q, draw a^{ϵ_q} from U to the endpoint of u, which we mark with E. Now, a geodesic v has the form $t^{-k}v_1v_2t^{-s}$ by the previous lemma, where v_1, v_2 are both type P, and v_1 has t-exponent q, and v_2 exponent $s \geq 0$. So v_1 is a geodesic from the point S to some point on level q of type P, so without loss of generality it is one of g_1, g_2 followed by some word a^ρ. So a geodesic for u is of the form

$$v = t^{-k}g_x a^{\rho_0} t a^{\rho_1} t \dots t a^{\rho_s} t^{-s}$$

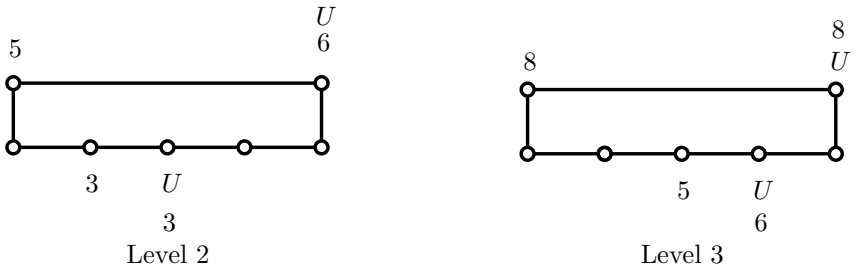

FIGURE 3. The next two levels

for $x = 1, 2$ and $s \geq 0$, and again $|\rho_i| < p$ for $i < s$ and $0 < |\rho_s| < 3p$. Now $a^{\rho_0} t a^{\rho_1} t \dots t a^{\rho_s} t^{-s}$ has length $\geq 2s + 1$ is a geodesic from d_x to E and this distance is at most $3p$ since $d(E, U) < 3p$ and $d(U, d_x) \leq 1$. Therefore $s \leq (3p - 1)/2$ is bounded, and so finding v is a matter of checking a finite number of possible suffixes, which can be done in constant time, so we are done. (Note that a much sharper bound could be obtained, but $s \leq (3p-1)/2$

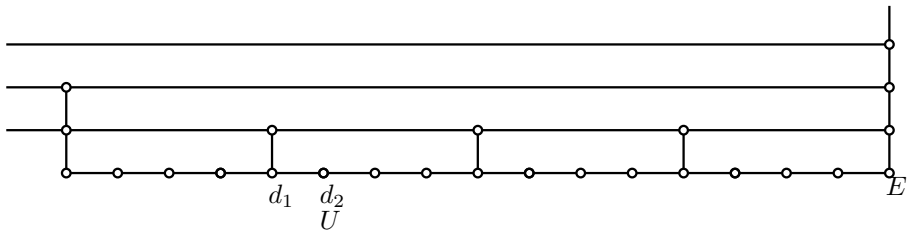

FIGURE 4. The top level

suffices to complete the proof). The final step then also takes constant time and space so all together we have used $O(q)$ time and space with $q \leq n$.

It follows that this v $3p$-fellow travels u asynchronously, since at each level U is within 1 of d_1, d_2, and so any point on u is contained in the same brick as some part of v up to level q, and at level q the suffix of v of length $\leq (3p-1)/2$ and the suffix of u of length $< 3p$, both paths from U to E, mean that any point on one is within $3p$ of the other. □

Finally we must deal with the cases when when u is type PN and NPN. The difference here is that the word v may not go up as high as u. As an instructive example, suppose $u = (a^{1-p}t)^n a t^{-n}$ $(p \geq 2)$, which is in the form out the output of part one of the algorithm. This word has geodesic

representative a, and so the geodesic for it no longer fellow travels it. In spite of this we have the following.

Lemma 7. *Let u be as in the previous lemma with $m > 0$, so this time, $u = t^{-k}a^{\epsilon_0}t \dots ta^{\epsilon_q}t^{-m}$ Then v can be found, as in the previous lemma, so that u, v asynchronously $3p$-fellow travel up to level $q - m$ in the sheet, in linear time and space in the length of u.*

Proof. Compute $q - m = c$, and write

$$u = t^{-k}a^{\epsilon_0}t \dots ta^{\epsilon_{c-1}}ta^{\eta_1}t \dots ta^{\eta_m}t^{-m}$$

(this can be done on the tape from part one, just find the column where the t-exponent is the same as the final t-exponent). Repeat the procedure from the previous lemma, up to level c. So we have a line at level c with points marked $U, d_1, (d_2)$. Now the geodesic v must come up to this level, and then end with a word of the form a^i or v_2t^{-s} with v_2 of type P. Note that $|i| < 3p$ or v is not geodesic. So extend the line at level c by $3p - 1$ edges to the right and left of the point(s) d_1, d_2. See Figure 5. Read a^{η_1} along the line from

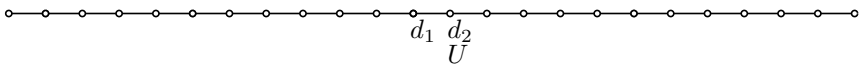

FIGURE 5. Level c, with $3p - 1 = 11$ edges on either side of d_1, d_2 added.

the point U, draw vertical t edge up, and cover the line with bricks as before. Store this data for the remainder of the procedure, since it may turn out that the shortest path runs along this line to the endpoint.

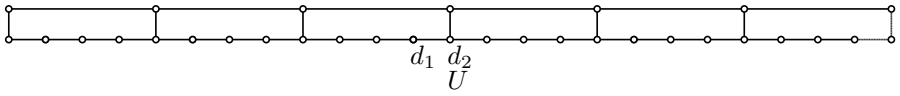

FIGURE 6. Covering level c with bricks. Here $\eta_1 = 0$.

Carry on as we did before, reading u and drawing bricks at each level, but this time extend out each line by $3p-1$ a edges in each direction from $d_1, (d_2)$. Continue until level $q = m + c$. To store this diagram of bricks takes $O(n)$ space, since for each level we are keeping a constant amount of data (at most 7 bricks at each level).

Read $a^{\epsilon_q} = a^{\eta_m}$ and mark this point by E_0. As before, if v extends above this level, then it can go only a bounded number of levels more, so draw these

layers of bricks in, aligned with the point E_0. So a geodesic to E_0 will be one of a finite number of paths, as before. Choose a shortest path to E_0, append t^{-m} to it, and store it with its length.

Consider the path t^{-m} from E_0 to E. Suppose each point is labeled E_i, with $E_m = E$. If a geodesic v for u does not travel up to level q, then it must travel to some level between q and c, via a point E_i. So for each E_i we will find and record a shortest path which travels to $d_1, (d_2)$ on that level, then across at most $3p-1$ a-edges to E_i then down to E by t^{-1} edges. We find these paths as follows.

Remove the layer of bricks which contain the point E_{i-1} from the stored diagram. If the point(s) $d_1, (d_2)$ on the level containing E_i is more than $3p$ edge(s) away then you have disconnected the diagram, and we know that a geodesic to E via E_i is not possible, since it would contain a subword of the form $a^{\pm 3p}$ which is not geodesic.

Otherwise record the path $t^{-k}g_x$ followed by a power of $a^{\pm 1}$ then t^{-1} edges, together with its length.

Do this for each level until you disconnect the diagram or reach the point E. Note that each level takes a constant amount of time and space, and once the diagram becomes disconnected, a lower level would not be connected again by the geometry of the sheet.

Therefore when you are done, search through the list of lengths recorded and choose a shortest, and output the geodesic path associated with it. □

References

[1] V. Diekert and J. Laun. On computing geodesics in Baumslag-Solitar groups. `http://www.arxiv.org/abs/0907.5114`.

[2] Murray Elder and Andrew Rechnitzer. Some geodesic problems in groups. `http://www.arxiv.org/abs/0907.3258`.

[3] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992.

[4] J. R. J. Groves. Minimal length normal forms for some soluble groups. *J. Pure Appl. Algebra*, 114(1):51–58, 1996.

[5] A. G. Miasnikov, V. Romankov, A. M. Vershik, and A. Ushakov. The word and geodesic problem in free solvable groups. `http://www.arxiv.org/abs/0807.1032`. To appear in Transactions of the American Mathematical Society.

[6] C. F. Miller III. Normal forms for some soluble Baumslag-Solitar groups, 1997. Unpublished.

School of Mathematics and Physics, University of Queensland, Brisbane, Australia

E-mail address, url: `murrayelder@gmail.com, http://sites.google.com/site/melderau/`